ECE4703 Laboratory Assignment 1

The goals of this laboratory assignment are:

- to familiarize you with the TMS320C6713 DSK hardware platform,
- to familiarize you with the process of creating, building, and testing some simple projects in the Code Composer Studio (CCS) integrated development environment (IDE),
- to familiarize you with some of the debugging, profiling, and visualization tools available within the CCS IDE, and
- to develop a code framework that you can re-use in the remaining laboratory assignments in ECE4703.

1 Preliminaries

This lab assignment is intended to familiarize you with most of the tools used to develop interesting real-time DSP systems in ECE4703. A portion of this assignment is self-study and is necessary to give you a better understanding of the capabilities of the hardware used in ECE4703. The remainder of this assignment is a hands-on exploration of the lab tools where you will develop three projects that will run on the TMS320C6713, interface with the basic I/O on the DSK, and perform some simple signal processing functions.

Please refer to the course website and/or syllabus for code and report due dates and times.

2 Self-Study: Familiarization with TMS320C6713 DSK

Begin by familiarizing yourself with the TMS320C6713 DSK technical reference (available as a PDF file on the course web page). You do not need to memorize everything in this document but you should become familiar with the key features and basic operation of the DSK. In Chapter 2, focus on the AIC23 codec (Section 2.2) and the LEDs and DIP switches (Section 2.5). Also be sure to check out the board layout and connector information in Chapter 3. You should know the function of the connectors on the DSK and where the main components are located.

3 Programming Assignments: Familiarization with CCS

The following three programming assignments will help you develop familiarity with Code Composer Studio including the development process and the debugging tools. Program 1 is the simplest possible program that you can run on the DSK and gives a gentle introduction to the development process. Program 2 is a framework for almost all of the signal processing algorithms you will develop later in the course. Program 3 is your first real-time DSP program, and builds on the framework developed in Program 2.

3.1 Program 1: Hello World

In this part of the assignment, you will compile and run the simplest possible program for the TMS320C6713 DSK. The goal is to simply generate the output "Hello World" on the CCS console. A "Hello World" template program is included with CCS.

If this is the first time CCS has been run on your computer, you may need to configure CCS to work with the DSK first. Please see this tutorial to set up a target configuration and connect to the DSK (the target configuration only needs to be set up once):

`http://spinlab.wpi.edu/courses/ece4703/configureccsv5.html`.

To create a new project from the helloworld template, follow the steps of the tutorial here:

`http://spinlab.wpi.edu/courses/ece4703/helloworld.html`.

When you build the helloworld project, it should compile with no errors or warnings. You might get some warnings about stack and heap sizes but the tutorial explains how to fix this. This program is the simplest possible program to get running on the DSK and ensures that you can successfully build, load, and run a program on the DSK without errors. This is the same process that we will be using for more complicated signal processing programs, so please make sure you play around with this program a bit to see what else you can do before proceeding. If you want to run the program again, what should you do? Try exiting CCS and then coming back in. How can you run the helloworld program now? Try printing "Hello World" 20 times in a loop. Try setting a breakpoint. Try profiling the execution time of the `printf` statement. Note that you may find the printf function handy in the future for debugging your code.

3.2 Program 2: Stereo Loop

In this part of the assignment, you will create a new project to run your first "signal processing" program on the DSK. This project will interface with the outside world through the AIC23 stereo codec. The AIC23 stereo codec is used for analog to digital as well as digital to analog conversion on the DSK. It is optimized for audio applications and is highly configurable. This program will simply read in samples from the left and right channels of the line input through the AIC23 and immediately output these samples (unchanged) through the AIC23 to the left and right channels of the line output.

Prior to creating the project, you should confirm you have the necessary chip support libraries and board support libraries on your computer. You may need to install these libraries yourself. Please follow the steps of the tutorial here:

`http://spinlab.wpi.edu/courses/ece4703/cslbsl.html`.

To create a new stereoloop project, follow the steps of the tutorial here:

`http://spinlab.wpi.edu/courses/ece4703/stereoloop.html`.

Here is some explanation for how this program works. We will discuss this in more detail in lecture, but here are the basics:

- This code

  ```
      DSK6713_init(); // Initialize the board support library, must be called first
      hCodec = DSK6713_AIC23_openCodec(0, &config); // open codec and get handle

      // Configure buffered serial ports for 32 bit operation
      // This allows transfer of both right and left channels in one read/write
      MCBSP_FSETS(SPCR1, RINTM, FRM);
      MCBSP_FSETS(SPCR1, XINTM, FRM);
      MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
      MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);

      // set codec sampling frequency
      DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_44KHZ);
  ```

  in `main()` calls functions in the DSK Board Support Library (BSL) to initialize the DSK, initialize the AIC23 codec, set it up for stereo operation, and sets the sampling frequency.

- The code uses an interrupt interface for dealing with the AIC23 codec. This is typically the most efficient way to interface to events that occur infrequently with respect to the clock of the DSP. This code sets up the interrupt interface:

  ```
      // interrupt setup
      IRQ_globalDisable(); // Globally disables interrupts
      IRQ_nmiEnable(); // Enables the NMI interrupt
      IRQ_map(IRQ_EVT_RINT1,15); // Maps an event to a physical interrupt
      IRQ_enable(IRQ_EVT_RINT1); // Enables the event
      IRQ_globalEnable(); // Globally enables interrupts
  ```

  Note that this code "hooks" the interrupt to the interrupt service routine (ISR) called "serialPortRcvISR". This is done in `main()` prior to the `while` loop and also in the file "vectors.asm". The initialization code in `main()` maps physical interrupt event (`IRQ_EVT_RINT1`) to an interrupt number (INT15) and enables interrupts. The file vectors.asm has some assembly language code that causes a branch to serialPortRcvISR to occur whenever INT15 is detected.

- Note the while loop at the end of `main()`. The DSP is sitting in an infinite loop and basically doing nothing unless it gets an INT15 signal.

- In the ISR,

  ```
  interrupt void serialPortRcvISR()
  {
      union {Uint32 combo; short channel[2];} temp;

      temp.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);
      // Note that right channel is in temp.channel[0]
  ```

```
        // Note that left channel is in temp.channel[1]

        MCBSP_write(DSK6713_AIC23_DATAHANDLE, temp.combo);
    }
```

the codec is accessed through the `MCBSP_read` and `MCBSP_write` functions.

- The tricky part here is how to interpret the results returned by the codec. The AIC23 codec returns both channels simultaneously in a 32-bit container, which we store as a Uint32 data type. The first 16 bits of this Uint32 variable are a signed integer representing the sample of one channel and the second 16 are a signed integer for the other channel. A convenient way to separate the channels is to use a "union" in C, e.g.

  ```
  union {Uint32 combo; short channel[2];} temp;
  ```

  With this structure, you can access the individual channel samples via `temp.channel[0]` and `temp.channel[1]`, and you can access the entire 32 bit chunk containing both channels with `temp.combo`.

This program will serve as a template for almost all of the remaining assignments in ECE4703, so it is important to understand what is going on and to get it working correctly. You may not fully understand all of the initialization details at this point, but we will cover this in more detail in an upcoming lecture. To ensure your code is working correctly, you may want to generate a test signal in Matlab with different signals in the left and right channels and listen to this test signal with headphones. You might want to also try swapping the channels in your code and testing that works as expected. Make sure this program is working before proceeding to Program 3.

3.3 Program 3: Squaring the Left Channel

This program will build on the stereo loop program above by reading both channels of the AIC23 and then outputting the channels to the AIC23 after squaring the left channel (the right channel will be passed through unmodified). Here are some suggestions on how to achieve this functionality:

1. Squaring the samples on the left input channel is a little bit more complicated than simply multiplying the sample by itself because you are likely to get overflow in the 16-bit signed integer datatype (with range -32768 to +32767) if you do this directly. Instead, consider the processing steps shown in Figure 1. By pre-scaling the samples (and converting to floats), the samples remain in the range $[-1, +1]$ after squaring. Unscaling and conversion to signed integers is necessary before writing the modified samples to the AIC23 codec.

2. Pass the right input channel samples directly to the right output channel.

3. Your code should compile without any errors or warnings. Load the code to the DSK and run it. Your code should work correctly and should be liberally commented.

4. Test your "signal-squarer" by sending a 1 kHz sinusoidal signal to both channels on the DSK and recording the output for analysis in Matlab. Note that there may be some strange things about the recorded signal. Is the recorded signal always positive (as you would expect from a squared quantity)? Does the recorded signal match perfectly with a squared sinusoid in

Matlab? You may want to think about the squaring operation in the frequency domain to explain what is going on. Careful analysis of the recorded signal will reveal some interesting properties of the AIC23 codec and its circuitry on the DSK.

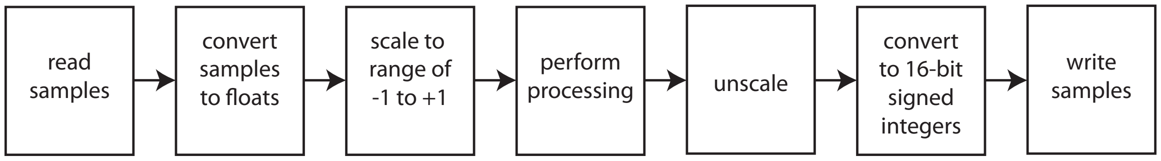
read samples
convert samples to floats
scale to range of -1 to +1
perform processing
unscale
convert to 16-bit signed integers
write samples

Figure 1: Typical steps when processing samples from the AIC23 codec

4 Additional Exploration

1. Even if you get your code working perfectly, it is important that you explore the debugging and code profiling features of CCS. The number one reason for difficulty in ECE4703 is an inability to effectively debug problems with your code. Make sure that you can:

   - Set breakpoints: It is very easy to set breakpoints in CCS. Just double-click in the gray left margin of the line where you want your code to stop. The breakpoint is indicated by a green circle with a checkmark in the left margin. Note that the execution stops *before* this line is run. You can clear the breakpoint in the same way by double-clicking the left margin.
   - Watch variables: This is also fairly straightforward. Use the CCS menu to select "View → Variables". The variables watch window will appear. To add a variable, click under the "Name" column and type the variable name. You can also specify how you want CCS to display the variable, e.g. decimal, hex, etc.

   Play around with CCS to better understand the full range of debugging features.

2. Make sure you are able to profile your code. This skill will be very important in later laboratory assignments when you need to ensure that your more complicated DSP algorithms are running in real-time. An interesting profiling experiment is to edit your linker command file to put all code and data for Program 3 in SRAM (your linker command file is probably already set up this way). Profile some code to determine the average number of cycles it takes for your code to complete. Then edit your linker command file to put all code and data for Program 3 in SDRAM. Profile the same code to determine the average number of cycles it takes for your code to complete. Is there any difference? Can you explain why or why not?

3. Finally, try out various signals in your "signal-squarer" including sinusoids with higher and lower frequencies, square waves, and even music or speech (you can use the computer's sound card output or bring in a portable audio player). Try listening to the output with a pair of headphones. Can you explain what is going on? Recall that multiplication in time domain is equivalent to convolution in the frequency domain.

5 In Lab

Teams of two or three are permitted. You will keep these lab partner(s) for all of the laboratory assignments in this course. You and your lab partner(s) will submit joint project code and lab reports that receive a single grade.

6 Specific Items to Discuss in Your Report

Please refer to the general report guidelines provided on the course web page for an overview of the ECE4703 report format. Since the first two programs in this assignment were quite simple and used code written by others, you do not need to discuss them in the report. Your report should focus on your methods, solutions, and results obtained with Program 3. To help understand what is going on with the "signal-squarer", you should also perform a few simple experiments with a function generator and an oscilloscope and report on the following characteristics of the AIC23 codec:

1. What is the maximum peak-to-peak input voltage (at the "line in" input) that the AIC23 codec can accept before clipping occurs?
2. What is the maximum peak-to-peak output voltage (at the "line out" output) the AIC23 codec can generate?
3. What happens when you try to sample a sinusoid with frequency higher than the Nyquist rate? To test this, set the sampling frequency of the codec to 44.1kHz (it should be set to this value already if you use the provided stereoloop.c code) and use the function generator to provide a 25kHz sinusoidal input to the codec.
4. What happens when you sample a signal with DC offset? To test this, use the function generator and apply a one volt offset to a 1kHz sinusoidal input with a one volt peak-to-peak voltage. Can you explain what is going on here?
5. What happens when you try to output a signal with a DC offset? Try generating an output with a DC offset by adding a constant to each sample before writing the sample to the AIC23 codec. Look at the result on a scope. What happens?
6. Discuss your SRAM/SDRAM ISR profiling results.
7. Finally, determine where the codec configuration is written in your code and take a look at the AIC23 codec datasheet (a link is provided on the course web site). Notice that, among other configuration options, the codec has registers that allow for left/right line input channel volume control in 1.5dB steps. In Sections 3.2.1 and 3.2.3 of the datasheet, it is stated that the ADC and DAC each have a full-scale range of 1.0 VRMS. Using what you know about the codec configuration, can you explain how these specifications agree or disagree with the first and second results that you obtained regarding the maximum peak-to-peak input and output voltage?

Where appropriate, include plots and/or screenshots from the oscilloscope.

7 Final Remarks

Please be aware that each of the laboratory assignments in ECE4703 will require a significant investment in time and preparation if you expect to have a working system by the assignment's due date. This course is run in "open lab" mode where it is not expected that you will be able to complete the laboratory in the scheduled official lab time. It is in your best interest to plan ahead so that you can use the TA and instructor's office hours most efficiently.